US006247141B1

(12) **United States Patent**
Holmberg

(10) **Patent No.: US 6,247,141 B1**
(45) **Date of Patent: Jun. 12, 2001**

(54) **PROTOCOL FOR PROVIDING REPLICATED SERVERS IN A CLIENT-SERVER SYSTEM**

(75) Inventor: **Per Anders Holmberg**, Stockholm (SE)

(73) Assignee: **Telefonaktiebolaget LM Ericsson (publ)**, Stockholm (SE)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/159,771**

(22) Filed: **Sep. 24, 1998**

(51) **Int. Cl.**[7] **G06F 11/14**; H04L 29/02

(52) **U.S. Cl.** **714/2**; 714/4; 707/1; 709/203

(58) **Field of Search** 714/2, 4, 43, 56, 714/15, 20, 3, 48, 758, 807; 707/1, 10, 204; 709/203, 217, 212, 227, 101, 228, 219, 216; 711/162; 370/216

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,879,716	11/1989	McNally et al. .
5,005,122 *	4/1991	Griffin et al. .
5,307,481	4/1994	Shimazaki et al. .
5,434,994	7/1995	Shaheen et al. .
5,452,448	9/1995	Sakuraba et al. .
5,455,932	10/1995	Major et al. .
5,488,716	1/1996	Schneider et al. .
5,513,314	4/1996	Kandasamy et al. .
5,526,492	6/1996	Ishida .
5,566,297	10/1996	Devarakonda et al. .
5,581,753	12/1996	Terry et al. .
5,634,052 *	5/1997	Morris .
5,652,908 *	7/1997	Douglas et al. .
5,673,381	9/1997	Huai et al. .
5,696,895	12/1997	Hemphill et al. .
5,751,997	5/1998	Kullick et al. .
5,796,934	8/1998	Bhanot et al. .

FOREIGN PATENT DOCUMENTS

0838758A2	4/1998	(EP) .

OTHER PUBLICATIONS

Murthy Devarakonda, et al., "Server Recovery Using Naturally Replicated State: A Case Study," IBM Thomas J. Watson Research Center, Yorktown Hts, NY, IEEE Conference on Distributed Computing Systems, pp. 213–220, May 1995.

Kenneth P. Birman, "The Process Group Approach to Reliable Distributed Computing", *Reliable Distributed Computing with the Isis Toolkit*, pp. 27–57, ISBN 0–8186–5342–6), reprinted from *Communications of the ACM*, Dec. 1993.

Robbert Van Renesse, "Causal Controversy at Le Mont St.–Michel", *Reliable Distributed Computing with the Isis Toolkit*, pp. 58–67, (ISBN 0–8186–5342–6), reprinted from *ACM Operating Systems Review*, Apr. 1993.

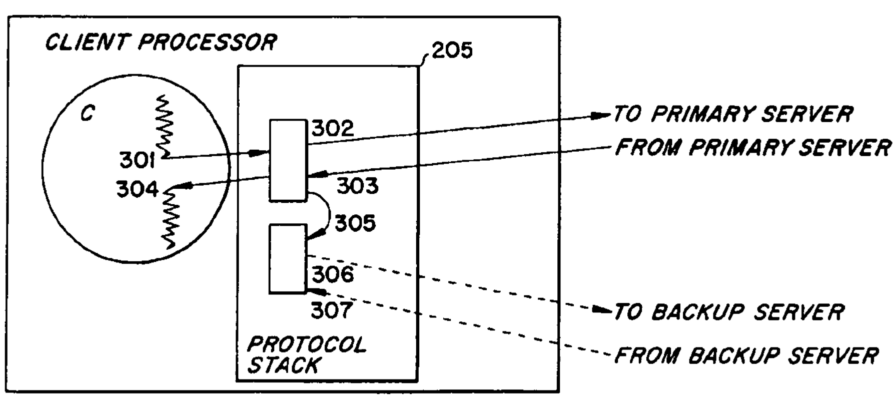
(List continued on next page.)

Primary Examiner—Gopal C. Ray
(74) *Attorney, Agent, or Firm*—Burns, Doane, Swecker & Mathis, L.L.P.

(57) **ABSTRACT**

A fault-tolerant client-server system has a primary server, a backup server; and a client. The client sends a request to the primary server, which receives and processes the request, including sending the response to the client, independent of any backup processing. The response includes the primary server state information. The primary server also performs backup processing that includes periodically sending the primary server state information to the backup server. The client receives the response from the primary server, and sends the primary server state information to the backup server. The primary server state information includes all request-reply pairs that the primary server has handled since a most recent transmission of primary server state information from the primary server to the backup server. The primary server's backup processing may be activated periodically based on a predetermined time interval. Alternatively, it may be activated when the primary server's memory for storing the primary server state information is filled to a predetermined amount.

10 Claims, 4 Drawing Sheets



OTHER PUBLICATIONS

Kenneth P. Birman, "Virtual Synchrony Model", *Reliable Distributed Computing with the Isis Toolkit*, pp. 101–106, (ISBN 0–8186–5342–6) 1994.

Carlos Almeida, et al. "High Availability in a Real–Time System", *Reliable Distributed Computing with the Isis Toolkit*, pp. 167–172, (ISBN 0–8186–5342–6), reprinted from *ACM Operating Systems Review*, Apr. 1993 and *Proceedings of the 5th ACM SIGOPS Workshop*, Sep. 1992.

Kenneth P. Birman, et al., "Reliable Communication in the Presence of Failures", *Reliable distributed Computing with the Isis Toolkit*, pp. 176–200, (ISBN 0–8186–5342–6), reprinted from *ACM Transaction on Computer Systems*, Feb. 1987.

Kenneth P. Birman, et al., "Lightweight Causal and Atomic Group Multicast", *Reliable Distributed Computing with the Isis Toolkit*, pp. 201.236, (ISBN 0–8186–5342–6), reprinted from *ACM Transactions on Computer Systems*, Aug. 1991.

Frank Schmuck, "Efficient Broadcast Primitives in Asynchronous Distributed Systems", *Reliable Distributed Computing with the Isis Toolkit*, pp. 263–283, (ISBN 0–8186–5342–6) 1994.

Timothy A. Clark, et al., "Using th Isis Resource Manager for Distributed, Fault–Tolerant Computing", *Reliable Distributed Computing with the Isis Toolkit*, pp. 300–308, (ISBN 0–8186–5342–6), reprinted from *Proceedings of the Twenty–Sixth Annual Hawaii International Conference on Systems Science*, 1993.

Dan Strassberg, "When Computers Must Not Fail . . . ", EDN, Aug. 17, 1995, pp. 42–50.

Inhwan Lee, et al., "Software Dependability in the Tandem Guardian System", IEEE Transactions on Software Engineering, vol. 21, No. 5, May 1995, pp. 455–467.

* cited by examiner

FIG. 1

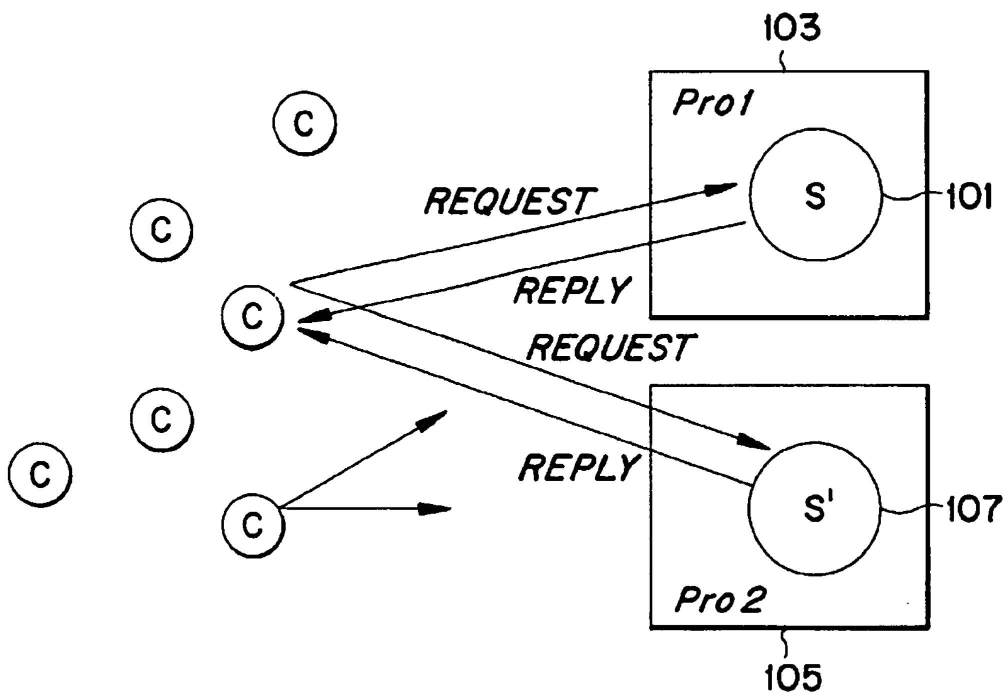

103
Pro1
S
101
REQUEST
REPLY
REQUEST
REPLY
C
C
C
C
C
C
S'
107
Pro2
105

FIG. 2

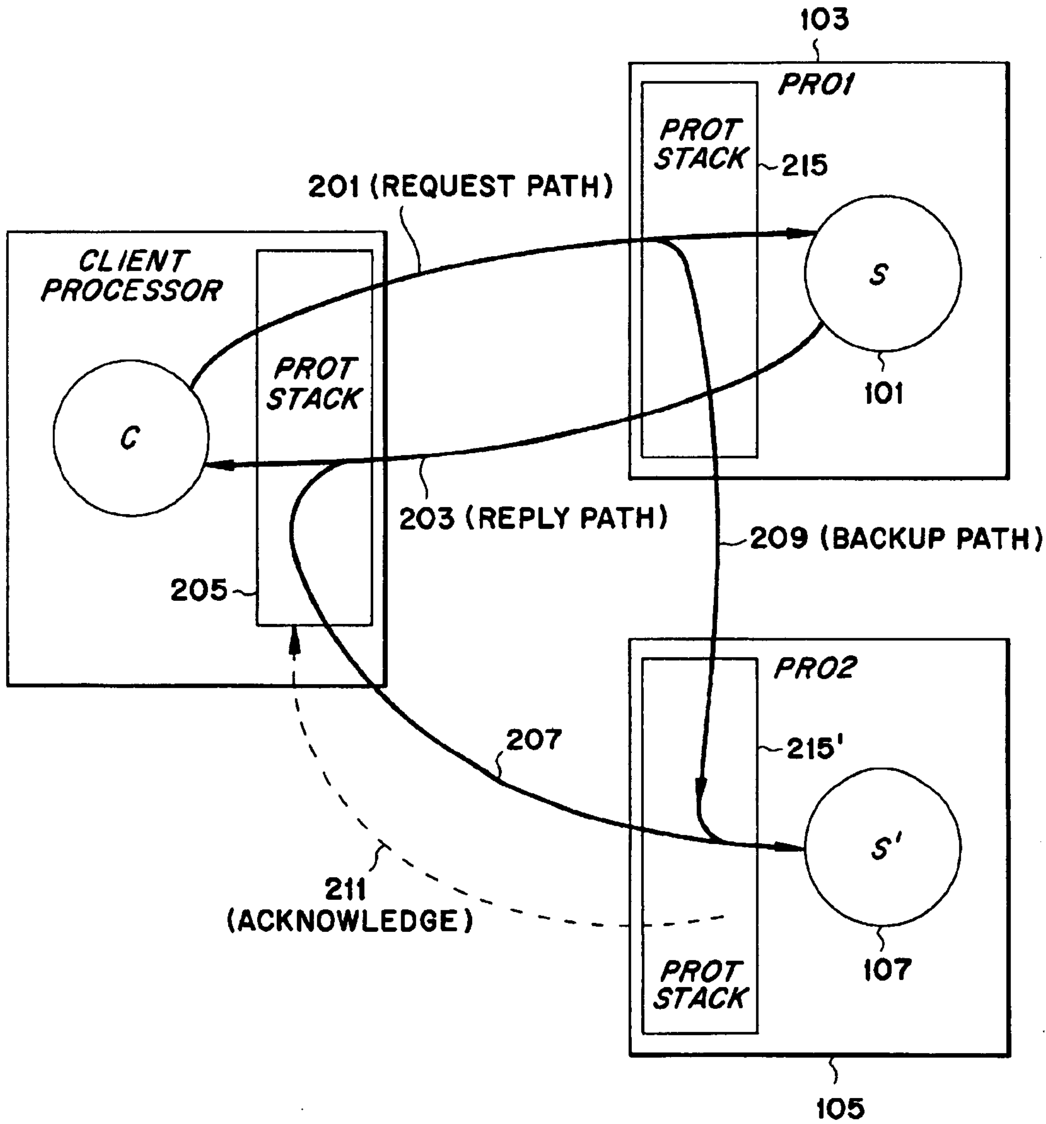
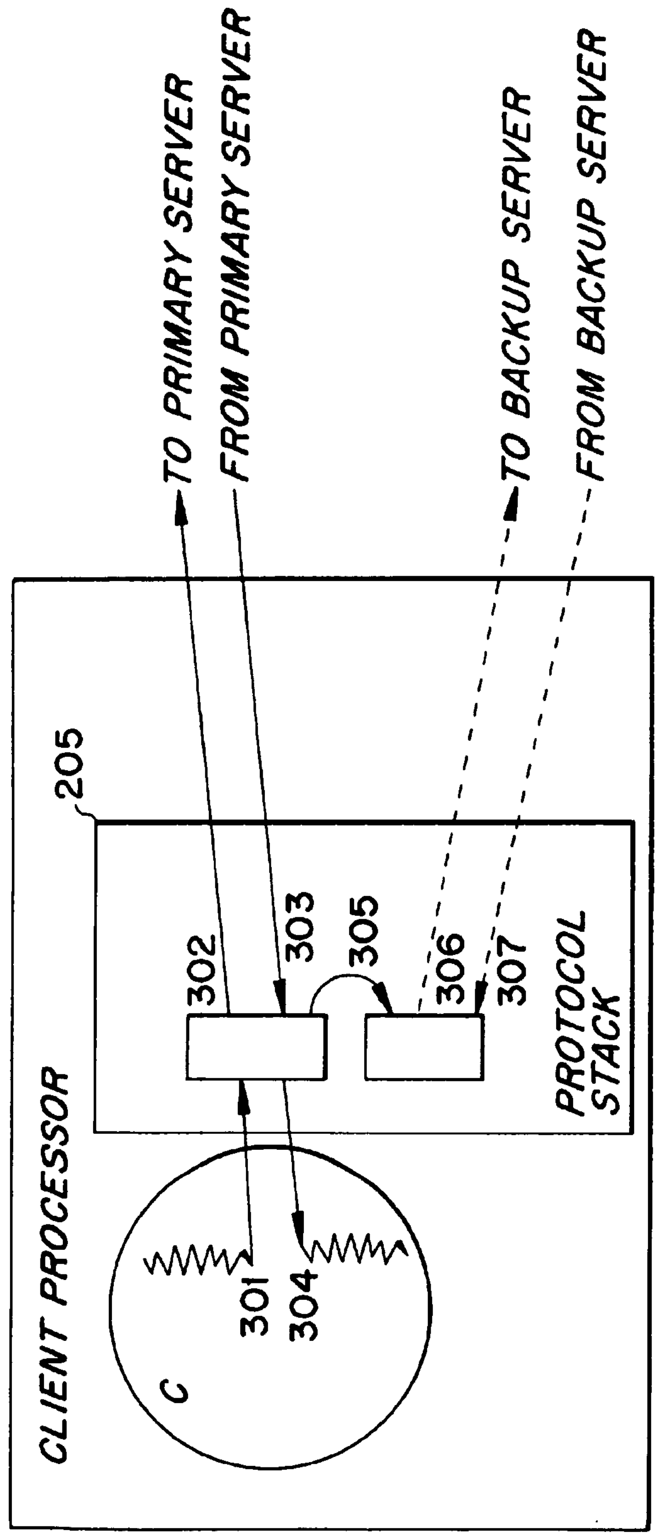

103
PRO1
PROT STACK
215
201 (REQUEST PATH)
CLIENT PROCESSOR
S
C
PROT STACK
101
203 (REPLY PATH)
209 (BACKUP PATH)
205
PRO2
207
215'
S'
211
(ACKNOWLEDGE)
PROT STACK
107
105

FIG. 3


TO PRIMARY SERVER
FROM PRIMARY SERVER
TO BACKUP SERVER
FROM BACKUP SERVER
205
302
303
305
306
307
PROTOCOL STACK
CLIENT PROCESSOR
C
301
304

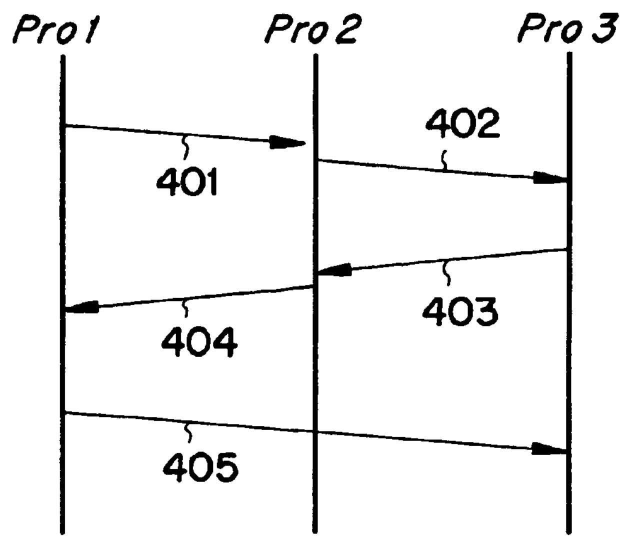

Pro 1
Pro 2
Pro 3
402
401
403
404
405

FIG. 4a

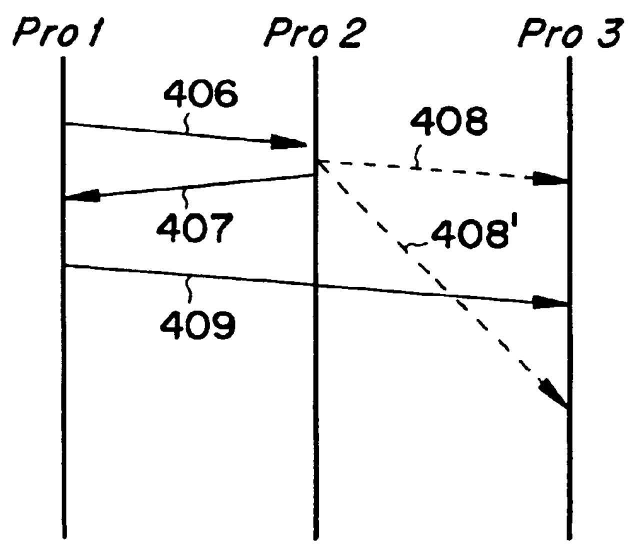

Pro 1
Pro 2
Pro 3
406
408
407
408'
409

FIG. 4b

PROTOCOL FOR PROVIDING REPLICATED SERVERS IN A CLIENT-SERVER SYSTEM

BACKGROUND

The invention relates to fault tolerant server systems, and more particularly to fault tolerant server systems including redundant servers.

High availability of service in a telecommunication system can be achieved by means of fault tolerant computers or distributed system architectures. The use of this redundancy, however, may adversely affect other system properties. For example, the utilization of redundancy on the hardware level increases cost, physical volume, power dissipation, fault rate, and the like. This makes it impossible to use multiple levels of redundancy within a system.

For example, distributed systems can incorporate replication between computers, in order to increase robustness. If each of these computers are fault tolerant, costs will multiply. Furthermore, if backup copies are kept in software, for the purpose of being able to recover from software faults, the cost of the extra memory will multiply with the cost of the fault tolerant hardware, and for the multiple copies in the distributed system. Thus, in order to keep costs low, it is advisable to avoid the use of multiple levels of redundancy. Since the consequence of such a design choice is that only one level of redundancy will be utilized, it should be selected so as to cover as many faults and other disturbances as possible.

Disturbances can be caused by hardware faults or software faults. Hardware faults may be characterized as either permanent or temporary. In each case, such faults may be covered by fault-tolerant computers. Given the rapid development of computer hardware, the total number of integrated circuits and/or devices in a system will continue to decrease, and each such integrated circuit and device will continue to improve in reliability. In total, hardware faults are not a dominating cause for system disturbances today, and will be even less so in the future. Consequently, it will be increasingly more difficult to justify having a separate redundancy, namely fault tolerant computers, just to handle potential hardware faults.

The same is not true with respect to software faults. The complexity of software continues to increase, and the requirement for shorter development time prevents this increasingly more complex software from being tested in all possible configurations, operation modes, and the like. Better test methods can be expected to fully debug normal cases. For faults that occur only in very special occasions, the so-called "Heisenbuggs", there is no expectation that it will be either possible or economical to perform a full test. Instead, these kinds of faults need to be covered by redundancy within the system.

A loosely coupled replication of processes can cover almost all hardware and software faults, including the temporary faults. As one example, it was reported in I. Lee and R. K. Iyer, "Software Dependability in the Tandem Guardian System," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 21, No. 5, May 1995 that checkpointing (i.e., the copying of a present state to a stand-by computer) and restarting (i.e., starting up execution from a last checkpointed state by, for example, reading a log of the transactions that have occurred since the last checkpoint and then starting to process new ones) covers somewhere between 75% and 96% of the software faults, even though the checkpointing scheme was designed into the system to cover hardware faults. The explanation given in the cited report is that software faults that are not identified during test are subtle and are triggered by very specific conditions. These conditions (e.g., memory state, timing, race conditions, etc.) did not reoccur in the backup process after it took over; consequently, the software fault does not reoccur.

A problem with replication in a network is that there are a few services, such as arbitration of central resources, that do not lend themselves to distribution. This type of service must be implemented in one process and needs, for performance reasons, to keep its data on its stack and heap. To achieve redundancy, this type of process must then be replicated within the distributed network. In a high performance telecommunication control system this replication must be done with very low overhead and without introducing any extra delays.

SUMMARY

It is therefore an object of the present invention to provide methods and apparatuses for implementing a fault-tolerant client-server system.

In accordance with one aspect of the present invention, the foregoing and other objects are achieved in a fault-tolerant client-server system that comprises a primary server, a backup server and a client. The client sends a request to the primary server. The primary server receives and processes the request, including sending a response to the client, independent of any backup processing being performed by the primary server, wherein the response includes primary server state information. By sending the response independent of backup processing, a higher level of concurrence is achieved, thereby making the system more efficient. The primary server also performs backup processing, including periodically sending the primary server state information to the backup server. The client receives the response from the primary server, and sends the primary server state information from the client to the backup processor.

In another aspect of the invention, the primary server state information includes all request-reply pairs that the primary server has handled since a most recent transmission of primary server state information from the primary server to the backup server.

In yet another aspect of the invention, the primary server stores the primary server state information in storage means. The act of performing backup processing in the primary server may be performed in response to the storage means being filled to a predetermined amount.

In an alternative embodiment, the act of performing backup processing in the primary server may be performed periodically based on a predetermined time interval.

BRIEF DESCRIPTION OF THE DRAWINGS

The objects and advantages of the invention will be understood by reading the following detailed description in conjunction with the drawings in which:

FIG. 1 is a block diagram that illustrates the use of redundant servers in a client-server application;

FIG. 2 is a diagram illustrating the message flow in a fault-tolerant client-server application;

FIG. 3 is a diagram illustrating the flow of messages between a client, a primary server and a backup server in accordance with one aspect of the invention; and

FIGS. 4*a* and 4*b* illustrate an efficiency improvement that is accomplished by means of the use of causal ordering in communications between processes.

DETAILED DESCRIPTION

The various features of the invention will now be described with respect to the figures, in which like parts are identified with the same reference characters.

FIG. 1 is a block diagram that illustrates the use of redundant servers in a client-server application. In particular, a plurality of client applications, C, are shown. A primary server, S **101**, runs on a first processor **103**. A second processor **105**, which is separate from the first processor **103**, runs a backup server, S' **107**, in parallel with the primary server S **101**. Overall, so that when one fails, the other can take over without any client application C noticing the problem, the primary server S **101** and the backup server S' **107** should have the same internal state at a virtual time, T, that occurs after processing any specific request from the client application C. (Since the backup server S' **107** trails the primary server S **101**, the backup server S' **107** reaches the virtual time later in real time than the primary server S **101** does.) The existence of replicated server processes should not be visible to the client applications C using the server. In order to implement such a strategy, the following problems need to be solved:

Addressing: The client application C should address the server in a consistent way, regardless of whether the service is being performed by the primary server S **101** or the backup server S' **107** (or both).

Replication and Incoming requests from different client applications C, as well

Synchronization: as fault and repair notifications, can arrive in different order to primary server S **101** and backup server S' **107** due to differences in the physical network between processors. However, these requests must be sorted in the same order.

Fault and Repair Server process failure and the start of a new server process

Notifications: must be detected by the server that is still working.

State Transfer: When a server process restarts after a failure, the working server must transfer its internal state to the new server before it can start processing requests.

In addressing the above problems, a preferred embodiment of the invention attempts to satisfy the following goals:

Solve the replication problem only once. The implementation of replication has many pitfalls and is complicated to verify. There are many possible faults that must be covered.

Add only a low overhead, and impose this only on communications to replicated processes.

Worst case response times during normal operation, in the case of failure, and also when reintegrating a new process should all be known in advance and kept to acceptable levels.

No extra messages should be added to critical timing paths. Many conventional implementation techniques violate this goal. For example, a primary server may have to send a message to the secondary server and get a reply back before sending a reply back to the client. It is desired to avoid this so that the system's real-time response times are not slowed down by the added redundancy.

Handle many clients and dynamic clients. Telecommunication applications typically have many possible clients for a server. This means that one cannot use algorithms that, for example, must update information in the clients when the server process fails or recovers. Also, client processes typically have short lifetimes (they may exist only during a call). This means that algorithms that require the server to keep track of clients cannot be used.

In order to make the protocol simpler, a preferred embodiment of the invention imposes several restrictions. Some of these restrictions can easily be lifted by making the protocol more general. However, their inclusion here facilitates a description of the underlying mechanisms involved. These restrictions are:

Only two servers are involved: a primary and a backup. It will be apparent to those of ordinary skill in the art that the protocol can be extended to include more.

Tolerance for one fault at a time, that is, a single client or server failure. The system must recover (for example by starting up a cold stand-by) before another fault can be tolerated.

Simple network configurations. Complicated network fault cases that, for example, split the network in two, with one of the server pairs in each, are not considered.

No large messages. Bulk data transfers and the like will probably overflow buffers or queues.

Soft real-time responses. In the normal case (i.e., without any malfunctioning server) it is possible to guarantee approximately the same response times as for systems utilizing non-replicated servers. However, longer response times must be accepted at the time of failure, recovery and reintegration. These longer response times can still be guaranteed not to exceed a predetermined maximum amount of time.

Deterministic operation of servers. As will be described in greater detail below, the backup server will receive periodic update messages from the primary server. The processing of these update messages in the backup server must be deterministic in order to guarantee that it will reach the same internal state as that of the primary when sending the update message. The server software cannot include non-deterministic system calls, such as calls to a time-of-day clock (which returns a different result, depending on when it is called), because such calls would cause the backup server to reach an internal state that differs from that of the primary server.

Thus, the state of the backup server must be 100% specified by the information that it receives from the primary server. This can be achieved in either of two ways:

a) the requests supplied to the primary server are also transferred to the backup server, which then reaches the same state as the primary server by doing identical processing of the request; or

b) the results of processing (i.e., the reply to the client that generated by the primary server, as well as the changes in the server's internal state) are sent to the backup server.

Simple applications only. In the description of the inventive protocol set forth below, the replicated server cannot request services from other servers. The protocol would have to be extended in order to handle such a case. In one such extension, the second server would then detect that a request comes from a replicated server and follow the same (or similar) protocol.

Earlier, four problems that need to be solved were mentioned. An inventive solution to one of these, namely replication and synchronization, will now be described. In a preferred embodiment, replication and synchronization are implemented as part of the communication protocol that is used between the client and the server. Advantages of this approach are:

The implementation is done only once, when the protocol is designed.

The replication is hidden from the application. The protocol handles addressing of the replicated servers.

The inventive protocol is designed for efficient implementation of the desired replication and synchronization:

1) Two alternative implementations are possible:
   a) The implementation may be an extension to the communication method. This means that there would be no extra system calls for processing a request from a client in the primary server.
   b) As an alternative, the protocol may be integrated into the protocol stack. This makes it possible to make more efficient implementations.

   So-called "middleware" solutions, in which fault tolerance is implemented by a layer of software on top of an existing operating system, would benefit from the first alternative (i.e., alternative "a") but not from the second (i.e., alternative "b").
2) The replication between servers can be outside the real-time critical loop. The client can get a reply as fast as the primary server S **101** can respond.
3) The extra information needed for keeping redundancy is attached to the reply in order to minimize overhead.
4) Updates/Heartbeats to the backup server S' **107** are done periodically in order to minimize overhead and to make it possible to guarantee that the recovery time after a fault will not exceed a predefined maximum. The number of requests that can be processed by the primary server but not by the backup server will be limited to the number of requests that can arrive between two periodic updates.
5) The replication can be supported within an I/O processor giving no overhead at all on the main processor.

The protocol guarantees that processed requests as well as information about the order in which the requests are processed, are always kept in two independent places in two separate computers. This strategy is based on two observations:

1) Redundant copies of the primary server state may be established at a later time than is conventionally performed, while still maintaining fault tolerance. That is, in conventional systems, server state information is transferred from the primary server to the backup server prior to sending a reply to the client. However, the invention recognizes that this is a conservative approach, because prior to sending a reply to the client, no other processor has seen the result. Consequently, a primary server crash would be considered to have occurred before the processing of the request. This is the case up to the time when the client receives the reply. This, then, is the latest possible time for establishing the existence of a redundant copy of the server state in order to have fault tolerance.
2) There are three independent parties involved: the client application C requesting a service, the primary server S **101**, and the backup server S' **107**. At any time it is sufficient that critical information be maintained in two redundant copies. However, these copies need not be maintained only by the primary server S **101** and the backup server S' **107** (as in a conventional two-phase commit protocol). Rather, the client can also be used for (temporarily) holding information.

For a simple server application, the replication is based on a message flow as illustrated in FIG. **2**. A client application, C, accesses a primary server **101** via a protocol stack **205** running in the client processor. Counterpart protocol stacks **215**, **215'** also run in the primary and backup server processors, PRO1 and PRO2. Requests **201** are sent from the client application C to the primary server S **101**. The protocol stack **215** of the primary server S **101** attaches a sequence number to the request and then processes the request. As a result of processing the request, the primary server S **101** generates and sends a reply message **203**, via the protocol stack **215**, to the client application C immediately. In accordance with one aspect of the invention, the server's protocol stack **215** performs the additional function of storing the incoming request **201** in a queue whose contents are periodically communicated, via backup path **209**, to the protocol stack **215'** of the backup server S' **107**. In accordance with another aspect of the invention, the reply message **203** to the client C also includes information indicating at what point in a sequence of incoming requests (since the last flush) the client's request **201** was processed (i.e., the sequence number).

When the client application's protocol stack **205** receives the reply message **203**, it does two things: 1) it passes the reply message **203** to the client application C, and 2) it sends a message **207** that may contain, for example, the original request as well as the reply to backup server's protocol stack **215'**, which passes it to the backup server S' **107**. In some embodiments, the backup server's protocol stack **215'** may send an acknowledge message **211** to the client's protocol stack **205**, thereby confirming receipt of the client's message.

In addition to the backup server's receiving information from the client application's protocol stack **205**, whenever the queue in the primary server's protocol stack **215** reaches a predetermined value, or alternatively when a predetermined amount of time has elapsed, the queue in the primary server's protocol stack **215** is flushed to the backup server S' **107** via backup path **209**. In addition to supplying the vital redundant information to the backup server S' **107**, the act of flushing also serves as a heartbeat, indicating to the backup S' **107** that the primary server S **101** is still alive. The time between flushes/heartbeats sets the maximum time for recovery when there is a fault.

The backup server S' **107** takes over execution when it fails to receive one or more heartbeats from the primary server S **101** and starts receiving requests from clients C.

The information that should be passed on to the backup server in order to guarantee that recovery is possible is: a) the original request, and b) the sequence number that was appended in the reply message. With this information, the back-up will (after a crash) be able to sort the requests to be in the same order in which they were processed by the primary server and then perform identical processing. The same information may be passed to the backup server S' **107** from both the client application's protocol stack **205** and the primary server's protocol stack **215**, although in the case of information coming from the primary server's protocol stack **215**, the sequence number is of less importance because the copy of the incoming requests may typically be passed on in the order in which they were processed.

Passing the entire primary server reply message (including the sequence number) to the backup makes it possible for the backup server to improve fault detection. In addition to using the sequence number for sorting out message order, the backup server S' **107** can then also verify that it is in synchronization with the primary server by comparing its own reply to the one from the primary server S **101**. It should be noted, however, that it is sufficient to pass on a substitute for this information, such as a checksum of the reply, for this purpose as well.

For the fault detection purpose, the full reply information can be passed on from either source (i.e., via the client C or via periodic updates from the primary server S **101**) or from both. In one embodiment, the full reply information is passed only via the periodic updates from the primary server's protocol stack **215** in order to minimize the amount of information that has to go the longer path via the client's protocol stack **205**.

There are also several alternatives to appending the sequence information to reply messages in the text. One alternative is to just append the sequence number the request was processed in. Another alternative is to include the entire request sequence since the last periodic update. These alternatives serve the same purpose, and each can be regarded as "server state information" because they each define the order of the actions that the backup server S' **107** must take in order to achieve an identical state as that of the primary server S **101**.

A number of fault cases, and how the invention handles them, will now be described:

Primary Server Crash Before Reply is Sent

In this case, the client C will not receive an acknowledge (i.e., reply message **203**) from the primary server S **101**. In response, the protocol stack **205** of the client C re-transmits the original request **201** to both the primary and secondary servers S **101**, S' **107**. Otherwise (i.e., in non-fault cases), the client application C sends the requests only to the primary server S **101**. (It should be noted that the client application is generally unaware of this fault tolerance-related activity, since it addresses only the single logical server. Address translation and communication to the two servers, S **101** and S' **107**, are handled by the protocol stack **205** within the client processor.) If the secondary server S' **107** misses the heartbeats from the primary server S **101**, it takes over. Otherwise, it simply discards the request received from the client C.

Primary Server Crash after Sending a Reply but before Information is Flushed to Backup

The information needed for updating the backup server S' **107** to the state that existed when the last reply was sent can be retrieved from update messages supplied by the client's protocol stack **205**. Messages in the "reply path" from the primary server S **101** to the client C contain both the reply to the client application as well as the update information to the backup server S' **107**. The client application need receive only the reply information from the client C, not the additional update information. As shown in FIG. **2**, the update information is forwarded from the client's protocol stack **205** to the backup server S' **107** (via the backup server's protocol stack **215'**). This update information is the same information that the backup server S' **107** otherwise receives by means of the periodic updates that are directly communicated by the primary server S **101**. The cost of adding some extra information in an already existing message is small compared to having to send an extra message for it.

Client Crash after Sending Initial Request

In this case, the backup server S' **107** receives information for updating itself when the primary server flushes its queue.

Primary System Crash

The primary server S **101** as well as any clients executing in the same processor **103** will be lost. The backup server S' **107** executes remaining commands from the last flushed queue and then gets updated up to the point given by the last reply to a client that is executing outside of the primary server's processor **103**.

Message Loss

Messages that do not get an immediate acknowledgment are re-transmitted once or twice before the receiving process (or processor) is considered to be faulty.

The client's protocol stack **205** will now be described in greater detail with reference to FIG. **3**. At step **301**, client application execution causes a request to be sent to the primary server. At step **302**, the request is processed in the protocol stack **205** and sent to the primary server. The protocol implements re-transmission at message loss and a copy of the message is kept for doing this. At step **303**, a reply has returned from the primary server. The reply is sent back to the client application processes without delay. A copy of the request and the associated reply are kept for the replication protocol. Because, in this example, the primary server is presumed to respond relatively quickly, there is no separate acknowledgment sent from the primary server to the client. That is, the reply that is returned from the primary server is sufficient to function as a reply. In other embodiments that include a relatively slow primary server, it may be necessary for the protocol to include a separate acknowledgment that would be sent from the primary server to the client prior to transmission of the reply.

At step **304**, the application process can resume execution without waiting for the replication to be performed. At step **305**, the protocol stack **205** stores the request as well as the reply in a queue that is designated for requests that are not yet replicated to the backup server.

At step **306**, the client sends a message containing the original request as well as the reply to the backup server. In response, the backup server returns an acknowledgment (step **307**) to the client, in order to confirm safe receipt of the client's message. It will be noted that without the acknowledgment, the client would have no other way of knowing that its message had been received because no other reply is expected from the backup server.

Earlier, several other problems were mentioned, namely Fault and Repair Notifications and State Transfer. The inventive solutions to these problems will now be described.

With respect to Fault and Repair Notification, the communication between the primary and secondary server also functions as a heartbeat. If the secondary server does not get updated regularly, it waits long enough to receive any outstanding client timeouts and then takes over. When a server process restarts, it checks whether there is an active primary server.

Regarding State Transfer, this is used at the time of restarting a failed server. The state of the executing server must then be copied to the restarted one before they, again, can work as a primary/backup pair. There is no fundamental difference between this state transfer and the type of state transfer needed when doing system software and hardware upgrades. Also, given the low number of hardware failures in modern processors, the state transfer mechanisms should be optimized for system upgrades.

It will be recalled that one aspect of the invention is a requirement that requests from different clients, as well as fault and repair notifications, must be sorted in the same order, even though they may arrive in different orders in the primary and backup servers S **101** and S' **107**. Thus, in some embodiments it may be beneficial to provide a mechanism for enforcing causal dependency (also referred to herein as "causal ordering") between messages. Essentially, this refers to the processing of messages in the order in which they had been logically issued, rather than in the strict order in which they may have been received. A more complete description of causal ordering may be found in connection with a description of the ISIS tool-kit, which was developed by Cornell University in Ithaca, New York, USA. The description may be found in K. P. Birman and R. van Renesse, "Reliable Distributed Computing with the ISIS toolkit,"

published 1994 by IEEE COMPUTER SOCIETY PRESS, ISBN 0-81865342-6. Causal ordering can be implemented with low overhead and can improve system efficiency by allowing a higher degree of concurrence. FIGS. 4a and 4b illustrate this efficiency improvement. In FIG. 4a, a processor Pro1 sends a request for a resource to a resource handler Pro2 (step **401**). Without support for causal ordering in the underlying system, Pro2 must send a message to the resource Pro3 to initialize it (step **402**). After the resource has replied that it is ready (step **403**), Pro2 is now permitted to send a reply to Pro1, informing it that the resource is available (step **404**). The processor Pro1 can now send a message to the resource Pro3 (step **405**). It will be observed that the behavior of each processor is constrained by restrictions designed to prevent one processor from receiving (and consequently processing) a later-sent message prior to an earlier-sent message.

Referring now to FIG. 4b, this illustrates an example in which the underlying system supports causal ordering. Again, the example begins with the processor Pro1 sending a request for a resource to a resource handler Pro2 (step **406**). Now, the resource handler Pro2 does not need to wait for a reply from Pro3. Instead, it immediately sends a reply to Pro1 informing that the resource is available (step **407**). At approximately the same time, Pro2 sends a message to the resource Pro3 to initialize it (step **408**). Because of this concurrence, the processor Pro1 is able to send its message to the resource Pro3 (step **409**) much sooner than in the example without causal ordering (FIG. 4a). This does not create any problems because the causal message ordering guarantees that Pro3 will process the initialization message before receiving the message from Pro1, even if the message from Pro2 gets delayed (alternative step **408'**).

It is not necessary to implement a full causal ordering model for the limited case in which clients call a replicated server, because in such cases the sequence number is sufficient to enable the replicated server to process requests in the proper order. However, the full model is called for when the protocol is extended to a more general case, such as to allow a replicated server to call another replicated server.

The invention has been described with reference to a particular embodiment. However, it will be readily apparent to those skilled in the art that it is possible to embody the invention in specific forms other than those of the preferred embodiment described above. This may be done without departing from the spirit of the invention. The preferred embodiment is merely illustrative and should not be considered restrictive in any way. The scope of the invention is given by the appended claims, rather than the preceding description, and all variations and equivalents which fall within the range of the claims are intended to be embraced therein.

What is claimed is:

1. A fault-tolerant client-server system, comprising:

a primary server;

a backup server; and

a client,

wherein:

the client comprises:

means for sending a request to the primary server;

means for receiving a response from the primary server,

wherein the response includes primary server state information;

means for sending the primary server state information to the backup server;

the primary server comprises:

means for receiving and processing the request;

means, responsive to the request, for sending the response to the client, independent of any backup processing, wherein the response includes the primary server state information;

means for performing backup processing that includes periodically sending the primary server state information to the backup server; and

the backup server comprises:

means for receiving the primary server state information from the primary server;

means for receiving the primary server state information from the client.

2. The fault-tolerant client-server system of claim **1**, wherein the primary server state information includes all request-reply pairs that the primary server has handled since a most recent transmission of primary server state information from the primary server to the backup server.

3. The fault-tolerant client-server system of claim **1**, wherein the primary server state information includes a checksum derived from a reply.

4. The fault-tolerant client-server system of claim **1**, wherein the primary server's means for performing backup processing is activated periodically based on a predetermined time interval.

5. The fault-tolerant client-server system of claim **1**, wherein:

the primary server further includes means for storing the primary server state information; and

the primary server's means for performing backup processing is activated in response to the means for storing the primary server state information being filled to a predetermined amount.

6. A method of operating a fault-tolerant client-server system that comprises a primary server, a backup server and a client, the method comprising the steps of:

sending a request from the client to the primary server;

in the primary server, receiving and processing the request, including sending a response to the client, independent of any backup processing being performed by the primary server, wherein the response includes primary server state information;

performing backup processing in the primary server, including periodically sending the primary server state information to the backup server;

in the client, receiving the response from the primary server; and

sending the primary server state information from the client to the backup processor.

7. The method of claim **6**, wherein the primary server state information includes all request-reply pairs that the primary server has handled since a most recent transmission of primary server state information from the primary server to the backup server.

8. The method of claim **6**, wherein the primary server state information includes a checksum derived from a reply.

9. The method of claim **6**, wherein the step of performing backup processing in the primary server is performed periodically based on a predetermined time interval.

10. The method of claim **6**, wherein:

the primary server further performs the step of storing the primary server state information in storage means; and

the step of performing backup processing in the primary server is performed in response to the storage means being filled to a predetermined amount.

* * * * *

US005887170A

United States Patent [19]

Ansberry et al.

[11] **Patent Number: 5,887,170**

[45] **Date of Patent: Mar. 23, 1999**

[54] **SYSTEM FOR CLASSIFYING AND SENDING SELECTIVE REQUESTS TO DIFFERENT PARTICIPANTS OF A COLLABORATIVE APPLICATION THEREBY ALLOWING CONCURRENT EXECUTION OF COLLABORATIVE AND NON-COLLABORATIVE APPLICATIONS**

[75] Inventors: **Catherine Malia Ansberry**, Redmond; **Jay Douglas Freer**, Bellevue; **Todd W. Fuqua**; **Erik Peter Mesterton**, both of Redmond, all of Wash.; **Catherine Ann Stillwagon**, Upper Saddle River, N.J.; **Ching-Yun Yang**, Austin, Tex.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **387,501**

[22] Filed: **Feb. 13, 1995**

[51] **Int. Cl.**^{6} **G06F 15/16**

[52] **U.S. Cl.** **395/687**; 395/200.35

[58] **Field of Search** 395/680, 200.35, 395/682

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,129,859	12/1978	Iwamura et al.	340/324
4,442,295	4/1984	Sukonick	364/521
4,533,910	8/1985	Sukonick et al.	340/721
4,642,790	2/1987	Minshull et al.	364/900
4,736,407	4/1988	Dumas	379/96
4,796,201	1/1989	Wake	364/518
4,831,556	5/1989	Oono	364/521
4,845,644	7/1989	Anthias et al.	364/521
4,860,217	8/1989	Sasaki et al.	364/518
4,893,326	1/1990	Duran et al.	379/53
4,937,856	6/1990	Natarajan	379/158
4,939,509	7/1990	Bartholomew et al.	340/717
4,953,159	8/1990	Hayden et al.	370/62
4,974,173	11/1990	Stefik et al.	364/521
5,003,532	3/1991	Ashida et al.	370/62
5,062,040	10/1991	Bishop et al.	364/200
5,065,347	11/1991	Pajak et al.	395/159
5,119,319	6/1992	Tanenbaum	364/514
5,148,154	9/1992	MacKay et al.	340/712
5,148,521	9/1992	Ebbers et al.	395/155

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

0 279 558 B1 2/1988 European Pat. Off. .

OTHER PUBLICATIONS

D. M. Chess et al., IBM Technical Disclosure Bulletin, vol. 30, No. 6, Nov. 1987.

P. A. Appino et al., IBM Technical Disclosure Bulletin, vol. 35, No. 4A, Sep. 1992.

S. P. Thompson, IBM Technical Disclosure Bulletin, vol. 36, No. 06B, Jun. 1993.

Nye Adrian, Xlib Programming Manual, O'Reilly & Assoc., pp. 75–76 Mar. 1993.

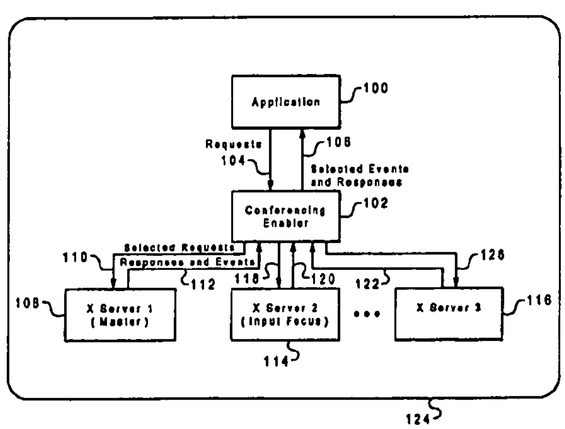
Nye, Adrian, Xlib Programming Manual, O'Reilly & Assoc., pp. 5–6,9–10 Mar. 1993.

Primary Examiner—Thomas C. Lee
Assistant Examiner—Albert Wang
Attorney, Agent, or Firm—Monica D. Lee; Andrew J. Dillon

[57] **ABSTRACT**

A method and system provide for selectively distributing communications between an application and multiple servers, allowing cooperative use of a single copy of an application. The system is situated between an application and the multiple servers. Requests from the application, responses to the requests, and events from the multiple servers, are managed in such a way that each server believes it is connected directly to the application and the application believes it is connected directly to a single server. The requests are categorized and distributed to the servers based on the type of request. The responses to these requests may be sent to the application or discarded based on the type of request and the role of the server sending the request. The events are also categorized and, based on the role of the server causing the event, they may be passed on to the application or discarded.

10 Claims, 3 Drawing Sheets



10/30/2003, EAST Version: 1.4.1

U.S. PATENT DOCUMENTS

5,175,854	12/1992	Cheung et al.	395/650
5,179,652	1/1993	Rozmanith et al.	395/155
5,191,644	3/1993	Takeda	395/158
5,195,086	3/1993	Baumgartner et al.	370/62
5,214,784	5/1993	Ward et al.	395/800
5,241,625	8/1993	Epard et al.	395/163
5,280,583	1/1994	Nakayama et al.	395/200
5,289,574	2/1994	Sawyer	395/157
5,293,619	3/1994	Dean	395/650
5,379,374	1/1995	Ishizaki et al.	395/155
5,392,400	2/1995	Berkowitz et al.	395/200
5,557,725	9/1996	Ansberry et al.	395/153

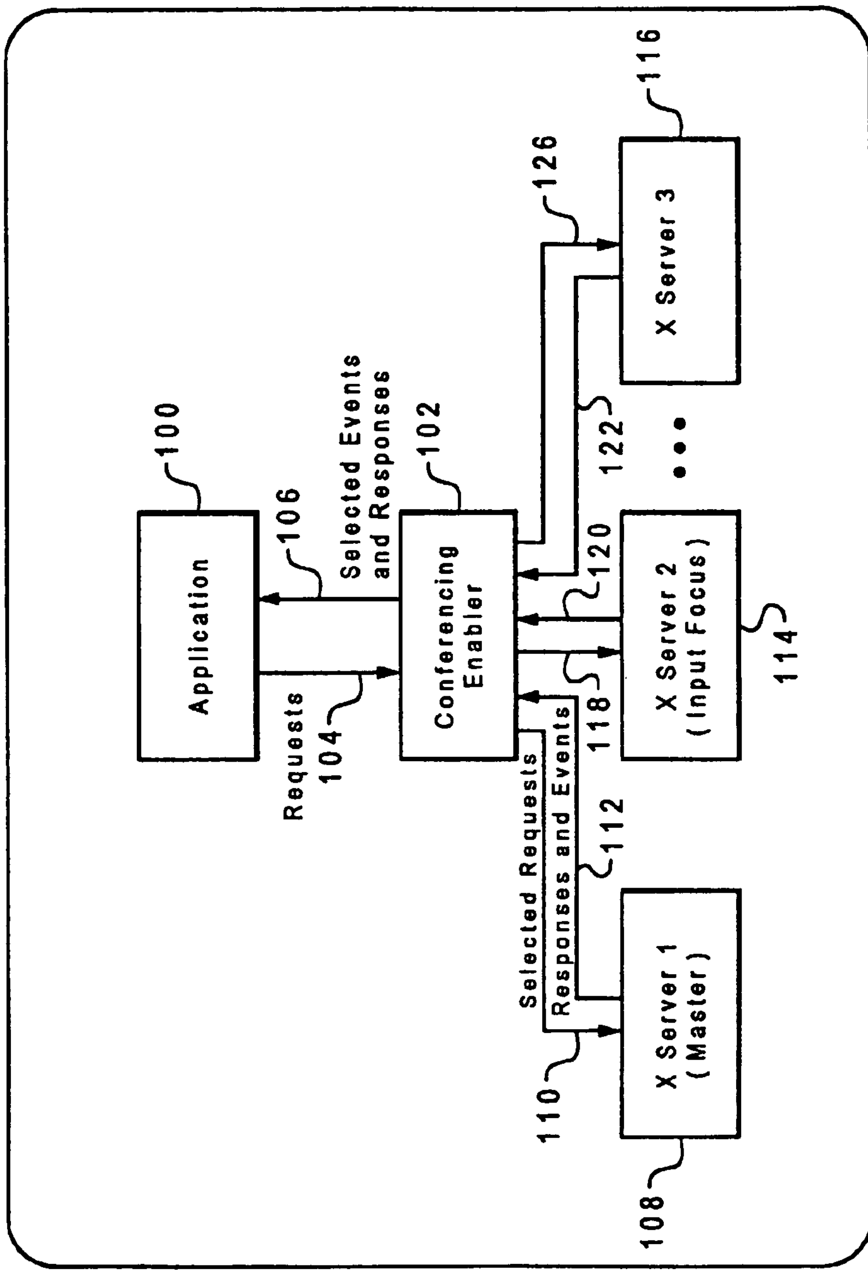

Application
100
Requests
104
106
Selected Events and Responses
Conferencing Enabler
102
Selected Requests
Responses and Events
110
112
X Server 1 (Master)
108
X Server 2 (Input Focus)
114
118
120
122
126
X Server 3
116
124

Fig. 1

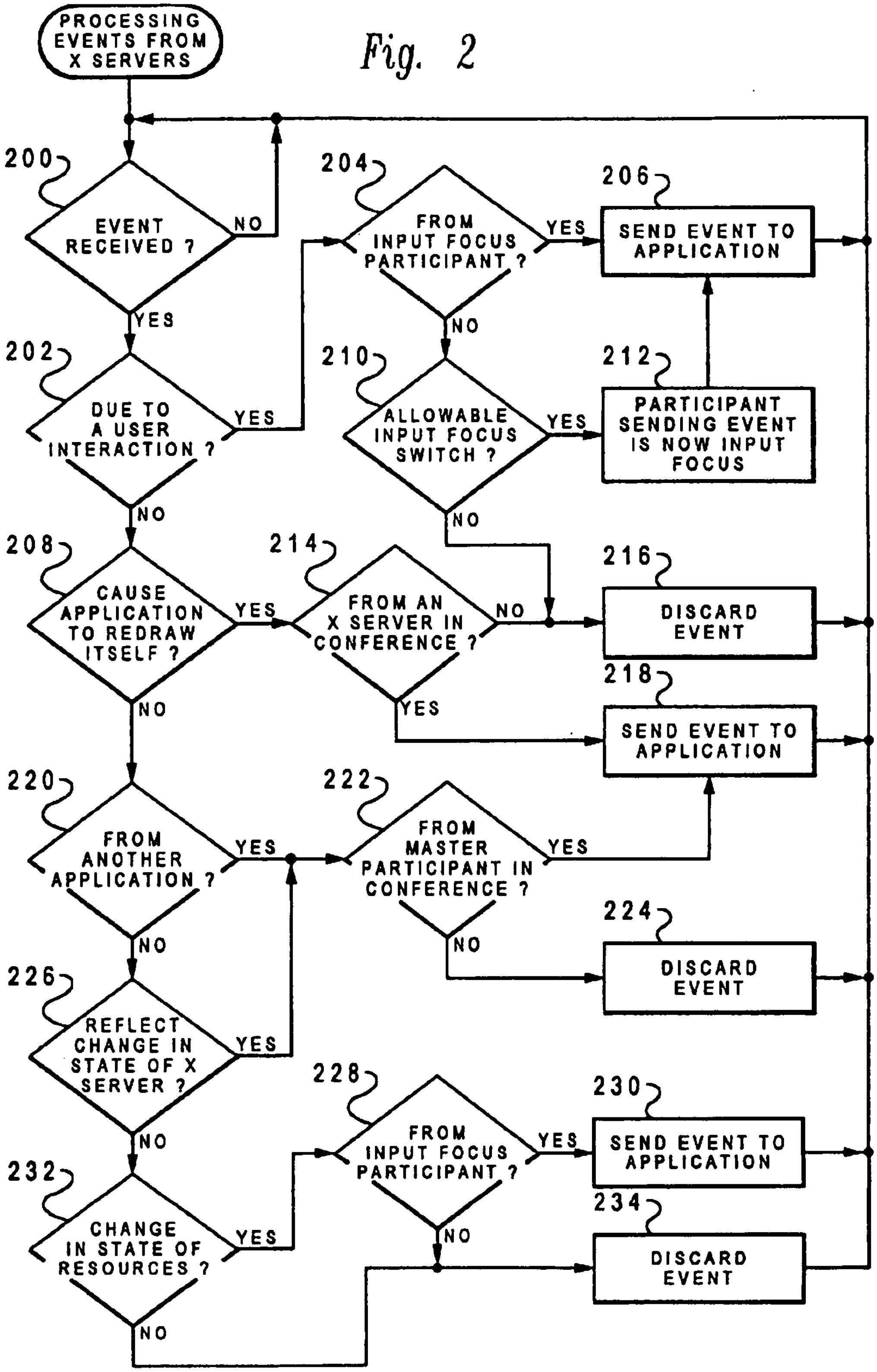

PROCESSING EVENTS FROM X SERVERS
200
EVENT RECEIVED ?
NO
YES
202
DUE TO A USER INTERACTION ?
YES
NO
204
FROM INPUT FOCUS PARTICIPANT ?
YES
NO
206
SEND EVENT TO APPLICATION
210
ALLOWABLE INPUT FOCUS SWITCH ?
YES
NO
212
PARTICIPANT SENDING EVENT IS NOW INPUT FOCUS
208
CAUSE APPLICATION TO REDRAW ITSELF ?
YES
NO
214
FROM AN X SERVER IN CONFERENCE ?
NO
YES
216
DISCARD EVENT
218
SEND EVENT TO APPLICATION
220
FROM ANOTHER APPLICATION ?
YES
NO
222
FROM MASTER PARTICIPANT IN CONFERENCE ?
YES
NO
224
DISCARD EVENT
226
REFLECT CHANGE IN STATE OF X SERVER ?
YES
NO
228
FROM INPUT FOCUS PARTICIPANT ?
YES
NO
230
SEND EVENT TO APPLICATION
232
CHANGE IN STATE OF RESOURCES ?
YES
NO
234
DISCARD EVENT

Fig. 2

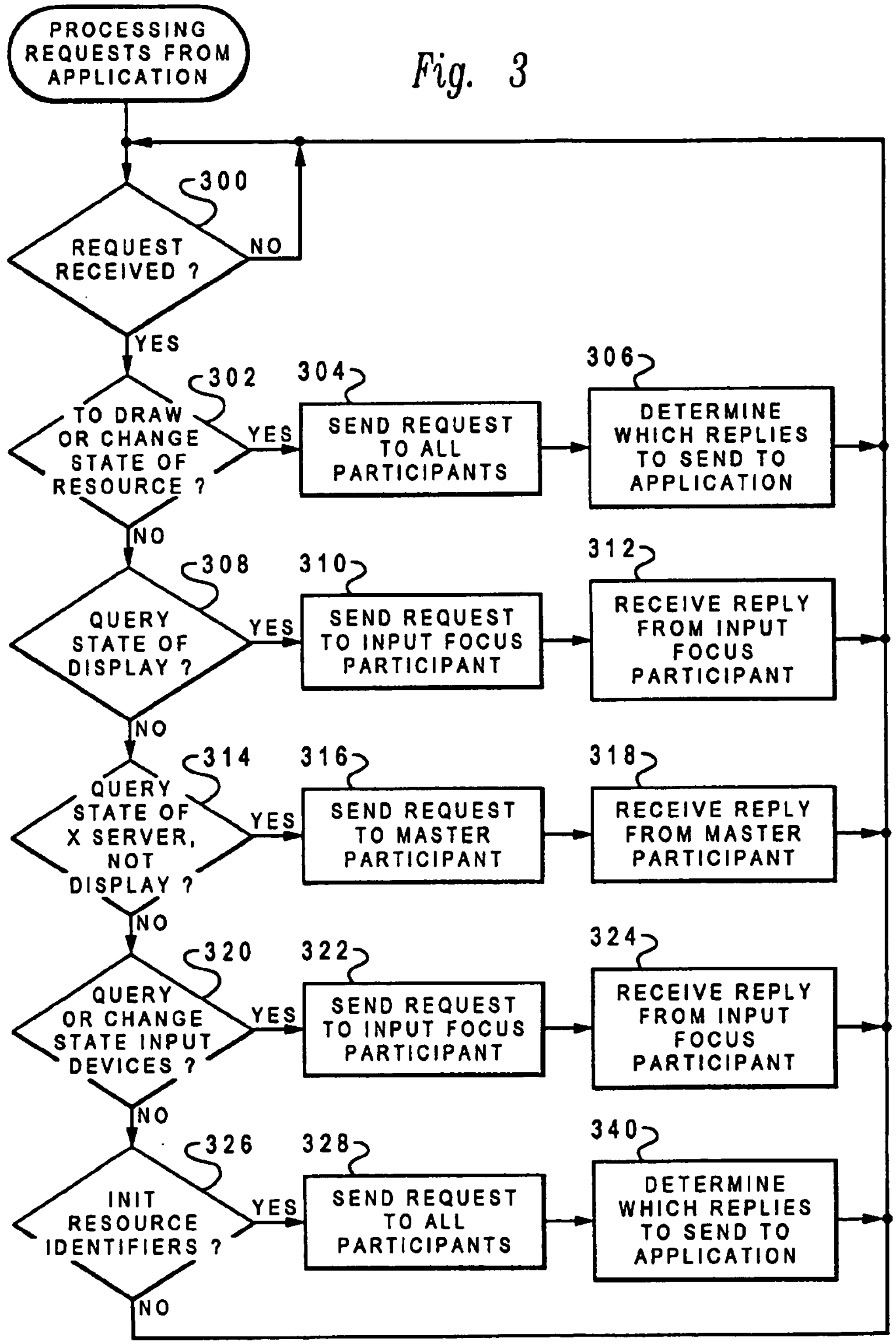

PROCESSING REQUESTS FROM APPLICATION
300
REQUEST RECEIVED ?
NO
YES
302
TO DRAW OR CHANGE STATE OF RESOURCE ?
YES
304
SEND REQUEST TO ALL PARTICIPANTS
306
DETERMINE WHICH REPLIES TO SEND TO APPLICATION
NO
308
QUERY STATE OF DISPLAY ?
YES
310
SEND REQUEST TO INPUT FOCUS PARTICIPANT
312
RECEIVE REPLY FROM INPUT FOCUS PARTICIPANT
NO
314
QUERY STATE OF X SERVER, NOT DISPLAY ?
YES
316
SEND REQUEST TO MASTER PARTICIPANT
318
RECEIVE REPLY FROM MASTER PARTICIPANT
NO
320
QUERY OR CHANGE STATE INPUT DEVICES ?
YES
322
SEND REQUEST TO INPUT FOCUS PARTICIPANT
324
RECEIVE REPLY FROM INPUT FOCUS PARTICIPANT
NO
326
INIT RESOURCE IDENTIFIERS ?
YES
328
SEND REQUEST TO ALL PARTICIPANTS
340
DETERMINE WHICH REPLIES TO SEND TO APPLICATION
NO

Fig. 3

SYSTEM FOR CLASSIFYING AND SENDING SELECTIVE REQUESTS TO DIFFERENT PARTICIPANTS OF A COLLABORATIVE APPLICATION THEREBY ALLOWING CONCURRENT EXECUTION OF COLLABORATIVE AND NON-COLLABORATIVE APPLICATIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is related to U.S. patent application Ser. No. 08/387,500, entitled Method and System For Switching Between Users In A Conference Enabled Application now U.S. Pat. No. 5,557,725, U.S. patent application Ser. No. 08/387,502, entitled Method for Managing Top-Level Windows Within a Conferencing Network System, U.S. patent application Ser. No. 08/387,503, entitled Method For Managing Visual Type Compatibility In A Conferencing Network System Having Heterogeneous Hardware now U.S. Pat. No. 5,715,392, U.S. patent application Ser. No. 08/387,504, entitled Method To Support Applications That Allocate Shareable Or Non-Shareable Colorcells In A Conferencing Network System Having A Heterogeneous Hardware Environment, U.S. patent application Ser. No. 08/387,505, entitled Method For Managing Pixel Selection In A Network Conferencing System, U.S. patent application Ser. No. 08/387,506, entitled Method And Apparatus For Translating Key Codes Between Servers Over A Conference Networking System now U.S. Pat. No. 5,640,540, all filed of even date herewith by the inventors hereof and assigned to the assignee herein, and incorporated by reference herein.

BACKGROUND OF THE INVENTION

1. Technical Field

The present invention relates in general to the field of data processing systems and in particular to the field of multiple user systems. Still more particularly, the present invention relates to the field of enabling multiple users to simultaneously use a single user application.

2. Description of the Related Art

The need to communicate as a group when the participants are not in the same room is becoming increasingly common. Past solutions include the use of faxes, teleconferencing and video conferencing. However, there have been few solutions for groups wanting to interact through a computer application. The participants can travel and meet in a single physical location, but the expense is often prohibitive. The participants could use a file sharing arrangement, but they would only be able to see their own session, not what the other participants are doing. Another approach is to allow everyone to see a view of one person's screen, but allow only the person with the application to interact with the application. Rewriting existing software to function in a multiuser mode is rarely a feasible solution.

The use of a conference has been proposed, and allows all participants to see the same working session. The use of this type of system allows pre-existing applications written for a single-user environment to be used from within the framework of a multi-user conference. A conferencing enabling module is located between the application and the users, and controls access to the application. These types of systems allow pre-existing applications to be used in a conference without the need for modifying the application. With this approach the problems which arise relate to handling input from multiple users.

The pre-existing applications are written with the assumption that they will be used by a single user. This assumption can cause some problems in a conferencing system because it leads to other assumptions. If there is only one user then the hardware of that user is not likely to change, the application is receiving only one stream of input, and the user wants communication from the application. These assumptions do not always hold true in a conferencing environment. The users in the conference may have different hardware, input may be sent from several workstations, and some users may be working on other applications, either in the conference or locally, and do not want communication from the application. The conferencing enabler must be able to handle outputs from and inputs to the applications and workstations so the application being conferenced is protected by giving the application the appearance of a single user environment.

U.S. Pat. No. 5,195,086, issued to AT&T Bell Laboratories, discloses a communication conferencing application which controls multiple concurrent calls sharing applications. This is effectuated by pseudo servers which control the flow of events from the servers to the application and necessary X resource identifier translation. This implementation only allows one party to input at a time. The method for controlling the flow of events for the X resource identifier translation is mentioned but not disclosed.

A system has also been disclosed by R. Wiss in the X Windows/MOTIF User Interface Server document. In this disclosure a system and method are described for the dynamic sharing of user interfaces which are coupled to applications, and a window management system provides concurrent event handling for multiple applications. This system and method controls the events from user interfaces of multiple applications.

U.S. Pat. No. 5,293,619, entitled "Method and Apparatus for Collaborative Use of Application Program", has a similar approach to the current invention. It discloses a single system, between an application and multiple X servers, which passes output requests from the application to the servers and passes events from the servers to the application.

One problem with the approach of the '619 patent is that, by passing requests to all servers, users cannot use applications other than the currently conferenced application. An example of the problem is when the conferenced application issues a request to freeze the keyboard while it is doing some type of activity. If this request is sent to all users, their keyboards will be frozen until the conferenced application has issued the unfreeze keyboard request, thereby inhibiting a user from doing any activity outside the conferenced application, such as checking E-mail.

Another problem with system described in the '619 patent is that by passing all events from the servers to the application, the program can get confused unless the users coordinate themselves well. An example of this problem occurs when two engineers are working on a CAD drawing of a widget and one engineer wants to move a part of the widget while the other engineer wants to delete the same part of the widget. Unless the two engineers coordinate their activities over the phone, the command to delete the part could be entered before the other engineer attempts to move the part and the program would try to move a nonexistent part. Unless there is sufficient error handling already in the application, the application could crash.

The usefulness of the '619 patent is also limited because it assumes the minimum common hardware for the servers. If all servers but one had a mouse, the conferencing system wouldn't be able to recognize mouse capabilities. If the user without the mouse just wanted to observe and not input, the

conference is still limited. This limits the capabilities of the system. This limitation could be even more of a problem if the hardware capabilities are mutually exclusive. The requirement that the workstations and applications be predetermined before invoking the conference, and can't be changed during the conference, limits the usefulness of the system of the '619 patent. If during the conference it is determined that different users or applications are either necessary or unnecessary, the conference must be closed and restarted with the new configuration.

Therefore, it will be apparent that a need exists for an improved method and system whereby a standardized system allows a single application to be distributed to multiple users in a conference to use concurrently without modification to the application.

SUMMARY OF THE INVENTION

It is therefore one object of the present invention to provide a system which allows multiple servers to use a single copy of an application.

It is another object of the present invention to allow each participant in the conference to interact with the application.

It is yet another object of the present invention to distribute application requests to one or more participants' server and manage the responses.

It is yet another object of the present invention to control events from each of the servers in the conference and decide which to send to the application.

The foregoing and other objects are achieved as is now described. A method and system provide for selectively distributing communications between an application and multiple users, allowing cooperative use of a single copy of an application. The system is situated between an application and the multiple servers running the application. Requests from an application, and the responses to them, from multiple servers running the single application, and events from multiple servers running the application, are managed in such a way that each server believes it is connected directly to the application and the application believes it is connected directly to a single server. The requests are categorized and distributed to the servers based on the type of request. The responses to these requests may be sent to the application or discarded based on the type of request and the role of the server sending the request. The events are also categorized and, based on the role of the server causing the event, they may be passed on to the application or discarded.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

FIG. 1 depicts a block diagram of a conference in accordance with a preferred embodiment of the present invention;

FIG. 2 depicts a logic flow illustrating event handling in the preferred embodiment of FIG. 1; and

FIG. 3 depicts a logic flow illustrating request handling in the preferred embodiment of FIG. 1.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

With reference to the figures and in particular with reference to FIG. 1, there is depicted a pictorial representation of the flow of information in the preferred system. The present invention is part of the conferencing enabler **102**. The conference enabler **102** comprises software that conceptually resides between an X Windows application and the X server. A conference is the shared use of an application by multiple users each having the same view of the program. The conference exists in an X Windows environment with multiple X servers. Those skilled in the art will recognize the variety of networks and workstations which could be used.

The conferencing enabler **102** is a program which runs continuously as a demon process in the background. It is conceptually situated between an application **100** and an X server. Its physical location is irrelevant, as it may actually be on the same workstation as the application, or on the X server, or at some other location. It is only required that both the X servers in the conference and the application have network access to the conferencing enabler. The conferencing enabler **102** appears to the application **100** to be an X server, while at the same time appearing to an X server to be an application. The conferencing enabler **102** then connects to multiple X servers on behalf of the application. Each participant in the conference may interact with the distributed application. The application **100** does not know that it is being distributed to multiple X servers. The conferencing enabler **102** determines how to multiplex and de-multiplex the requests from the application **100** and the replies, events and errors from the X servers in such a way that both the application **100** and the X servers are receiving meaningful information.

The conferencing enabler **102** receives requests **104** from an application **100** and distributes them among the servers **108**, **114**, **116** in the conference **124** associated with that application **100**. When a request is received, the conferencing enabler **102** must determine which X servers must receive the request. The conferencing enabler **102** also receives events and responses to requests **112**, **120**, **122** from the X servers **108**, **114**, **116** and determines which are sent **106** to the application **100**. The events from the X servers are managed in such a way as to present a consistent input stream for the application **100**.

The conference **124** begins by a user requesting a conference from the server, in this case X server **108**. Only certain users may start a conference. The X server which initiates the conference, in this case X server **108**, is given the role of master. The master controls the conference **124**. Only one X server can be the master in a conference and once the role is established it is irrevocable. If the master **108** leaves the conference **124**, the conference **124** is closed and all applications associated with that conference **124**, in this case **100**, are terminated. The first X server **108** is also initially assigned the role of input focus. There can only be one server assigned the role of input focus for each application **100** in the conference **124** at any one time, but this role can be switched among the X servers **108**, **114**, **116** in the conference by predetermined triggers.

For purposes of example, X server **114** is the input focus. The input focus **114** is the only server allowed to input to the application by key presses or button events. The X servers of the users in the conference that are not assigned the role of master **108** or input focus **114** have no specific role. These servers can display and manipulate the display of the applications but they can not input to the application without obtaining the input focus role. Because the application believes that it is connected to a single server, the role of master **108** represents the hardware of the single server and the input focus **114** represents the display and input of the single server. The aster **108** is not allowed to change because

the hardware of the X server attached to the application is not expected to change while the application is running. Limiting the input focus to one X server at a time ensures that the application has one view of the display and input from one X server. Otherwise the application is required to do something it wasn't designed to do.

After the first X server 108 has started the conference 124, other users may request to join the conference 124 from their servers, X server 114 and X server 116. With each request a small amount of code is place on the server regarding: members of the conference, who can join, who can launch an application, etc. These are typical user interface functions, known in the art, and will not be further described herein. After the conference 124 is established, members 108, 114, 116 of the conference 124 can leave, except the master 108, and others can be added. Once the master 108 has started the conference 124, certain X servers can launch an application 100 for the conference. More than one application 100 can be launched for a given conference 124 with the same copy of the conferencing enabler 102. The conference enabler 102 can distinguish the requests, responses and events for the different applications.

The invention takes the events 112, 120, 122 from the X servers 108, 114, 116 and determines which category they are a member of, and which X server they originated from, to determine whether they are sent to the application 100 or discarded. FIG. 2 is a flow representation of the logic involved in determining whether the events 112, 120, 122 are sent to the application 100. Referring to FIG. 2, when the conferencing enabler 102 receives 200 an event 112, 120, 122 from an X server 108, 114, 116, the type of event and the X server from which it originated determine whether the event is passed on to the application 100 from the X servers 108, 114, 116.

If the event 112, 120, 122 received 200 is of the type which is due to a user interaction 202, such as a key press, if (204) it is from the input focus 114 it is sent 206 to the application 100. If it is not from the input focus it is discarded 216 unless 210 it is an allowable input focus switch, in which case the participant sending the event is now the input focus 212 and 206 the event is sent to the application 100. The rationale for sending these type of events to the application 100 is that the input focus 114 is the only X server allowed to supply input to the application 100. This ensures that the application 100 does not receive an inconsistent sequence of events originating from multiple participants. The exception exists when the focus is to be changed.

If the event 112, 120, 122 received 200 is of the type which causes an application 100 to redraw itself 208 and 214 it is from an X server in the conference 124 it is 218 sent to the application 100, otherwise it is discarded 216. An example of this type of event occurs when a window which was previously covered becomes exposed. The rationale for sending these type of events to the application 100 if they are from any X server in the conference 124 is to ensure that the application 100 is displayed correctly on every X server. Since participants can interact independently with the conferenced application, taking these type of events only from one X server will not ensure the application is correctly displayed on every X server. If a user is working on a different application, their view of the conferenced application will be updated, but they will be otherwise unaffected.

If the event 112, 120, 122 of any type is received 200 from another application and it is 222 from the master 108 in the conference 124 it is 218 sent to the application 100, otherwise it is discarded 224. The rationale for sending these type of events to the application 100 only if they are from the master 108 in the conference 124 is because if this type of event is received it means that two applications are trying to communicate. If the other application, the one sending the event, is also part of the conference, then the event will only be received on the master and it should be sent on to the application. If the event is being sent from an application that is not part of the conference, then it is only sent if it is received on the master so as to continue the illusion that the master's X server is the only one connected to the application.

If the event 112, 120, 122 is received 200 is of the type which reflects a change in the state in the client's resources 232 and 228 it is from the input focus 114 for the application 100, it is 230 sent to the application 100, otherwise it is discarded 216. An example of this type of event is one which indicates the window has become mapped. The rationale for sending these type of events to the application 100 only if they are from the input focus 114 is because these type of events are most likely generated as a result of a request from the application 100 or from another client connected to the same X server and are most likely sent in response to a user interaction from the input focus 114. They are sent only from the input focus 114 to ensure the appearance of a single X server inputting to the application 100.

If the event 112, 120, 122 received 200 is of the type which reflects a change in the state of the X server 226 and it is from the master 108 in the conference 124, it is 230 sent to the application 100, otherwise it is discarded 234. An example of this type of event is one which indicates that the keyboard key mappings have changed. The rationale for sending these type of events to the application 100 only if they are from the master in the conference 124 is to ensure the application 100 is presented with a consistent X server. Because the application 100 believes it is connected to a single X server, the application 100 could become confused if state changes were forwarded from other servers.

The conferencing enabler receives the requests 104 from the application 100 and determines which type they are to determine which X servers receive the requests 104 and which responses to these requests to send back to the application 100. FIG. 3 is a representation of the logic involved in determining which X server 108, 114, 116 the requests 104 from the application 100 are sent to. When the conferencing enabler 102 receives 300 a request 104 from an application 100, the type of request 104 determines which X servers 108, 114, 116 the request is sent to and which replies to the request 104 are to be received by the application 100 from the X servers 108, 114, 116.

If the request 104 is of the type which is to draw geometry or change the state of a resources 302 then 304 the request 104 is sent to all X servers 108, 114, 116. An example of this type of request is one which asks to draw a line. The rationale for sending these type of requests to all of the X servers is so each X server maintains nearly identical states. It is important that everyone is able to view the same image. There are no replies to these type of requests, but an X server may respond with an error. If an error is received from an X server, generally they are discarded 306 unless they are from the master 108. The reason only the errors from the master 108 are passed back to the application 100 is because the errors will usually be due to something the application did wrong, so it is sufficient to receive the error from one X server and for consistency the master 108 is chosen.

If the request 104 is of the type which queries the state of the display 308 of an X server, then 310 the request 104 is

sent to the input focus 114. An example of this type of request is one which asks about the current size or contents of a window. The rationale for sending these type of requests only to the input focus 114 is that each X server can have significantly different views because they are allowed to minimize, cover or even close windows they are not specifically interested in. The input focus 114 should have a view which matches the application's perception of the state of the display, so it should be only the X server 114 which is allowed to provide input to the application 100 which supplies the state of the display. It is important that the application 100 believe it is attached to only one X server with only one display. The replies to these type of requests are received 312 only from the input focus 114 for the same reasons they are only sent to the input focus 114.

If the request 104 is of the type which queries the state of the X server 314, then 316 the request 104 is sent to the master 108. An example of this type of request is one which asks about the fonts available on the server. The rationale for sending these type of requests only to the master 108 is because it is important that the application 100 believe it is attached to only one X server 108, and the state of that X server 108 is not expected to change during the running of the application. The replies to these type of requests are received 318 only from the master 108 for the same reasons they are only sent to the master 108.

If the request 104 is of the type which queries or changes the state of input devices 320 of an X server, then 322 the request 104 is sent to the input focus 114. The rationale for sending these type of requests only to the input focus 114 is because each X server can have significantly different views. Because these requests may change the state of input devices the input focus 114 should be forced to match the application's 100 view of input devices while the other X servers need not be. An example of this type of request is a request which changes the location of the pointer. The input focus 114 should have the correct pointer location while other X server users may be working on a different application and their pointer should not be changed. It is important that the application 100 believe it is attached to only one X server with only one display. The replies to these type of requests are received 324 only from the input focus 114 for the same reasons they are only sent to the input focus 114.

If the request 104 is of the type which initializes resource identifiers 326 then 328 the request 104 is sent to all X servers 108, 114, 116. An example of this type of request is one which asks the server to create an atom. The rationale for sending these type of requests to all of the X servers is that the request creates a resource to be used in the future, so that resource must be created on all of the X servers in order for the subsequent request to be successful. Some of these type of requests may generate either replies or errors. Replies are sent from the master, simply so that the application receives resource identifiers that are in accord with those that exist on the master. Errors are returned from the master because an error from this type of request usually indicates that the application has done something wrong to generate that error. For consistency, therefore, the error is returned if it comes from the master.

Several conferences can utilize the conferencing enabler 102 at the same time. The conferences can be made up of different combinations of X servers. Each conference 124 has its own master 108 and each application within a conference 124 has its own input focus 114. Each conference 124 can have more than one application running.

A typical example of the use of this system and method is in a help desk situation. If a user is having a problem with an application they can conference with the helper. The user can show the helper where the problem is on the application and the helper can see the same view of the application the user is interacting with, without being in the same room looking at the display of the user's X server. The input focus can then be switched, allowing the helper to show the user how to fix the problem or the correct way to operate the application such that the user sees the helper's interaction with the application. During this session additional applications or people can be added to solve the problem. The user and helper communicate over the telephone during the conference to explain what they are doing and to coordinate the switching of the input focus.

Another example of the use of the invention is a team working on a project, each member of the team possibly having a workstation with a different hardware configuration than the others. The project leader begins the conference and the team joins the conference. While the engineers work together to design their widget, one of the engineers gets an E-mail message. She switches over to the E-mail to read her message while the other engineers continue designing and are unaffected. After their design is complete, one of the engineers brings up a simulation program to see how the new design will perform. All participants can see the simulation program. To interpret the data another engineer is added to the conference to see where problems exist in the new design as pointed out by the simulation program. Once the other engineers know what problems they need to fix, the simulation analyst leaves the conference and the conference drops the simulation program. The problem doesn't involve some of the engineers so they too may leave the conference.

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

What is claimed is:

1. A management system for executing collaborative and non-collaborative applications within a distributed computing environment, said system comprising:

at least one collaborative application;

at least one non-collaborative application;

a plurality of participants, wherein one participant is assigned the role of master, for controlling the system, and one participant is assigned the role of input focus, for inputting data to said at least one collaborative application;

an interface which facilitates communication between each of said plurality of participants and said at least one collaborative application; and

means within said interface for selectively classifying and distributing communication requests from said at least one collaborative application to said plurality of participants based upon categories of the classified requests, and responses and events from said plurality of participants to said at least one collaborative application wherein selected requests from said collaborative application are sent to all of said plurality of participants; wherein selected requests from said at least one collaborative application which query the state of a participant not directly related to a display are sent only to said master participant; and wherein requests from said collaborative application which query the state of input devices are sent only to said input focus participant so as to permit the manipulation and execution of said at least one non-collaborative application by all other participants.

2. The management system of claim 1, wherein said classified request categories include:

requests to draw or change the state of the operating system's resources, requests to query the state of the display, requests to query the state of a participant not directly related to the display, requests to query the state of input devices, and requests which initialize resource identifiers for said at least one collaborative application.

3. The management system of claim 2, wherein:

said requests from said at least one collaborative application which draw or change the state of the operating system's resources are sent to all of said plurality of participants;

said requests from said at least one collaborative application which query the state of the display are sent only to said input focus participant; and

said requests from said at least one collaborative application which initialize resource identifiers are sent to all of said plurality of participants.

4. The management system of claim 3, wherein selected responses from said plurality of participants to requests are sent to said at least one collaborative application based on the category of request to which said response is responsive.

5. The management system of claim 4, wherein:

responses to requests to draw or change the state of said plurality of participants' resources are sent only from said master participant;

responses to requests to query the state of the display are sent from said input focus participant;

responses to requests which query the state of said plurality of participants, not directly related to the display, are only sent from said master participant;

responses to requests which query the state of input devices are sent only from said input focus participant; and

responses to requests which initialize resource identifiers are sent only from said master participant.

6. The management system of claim 1, wherein said means for selectively classifying comprises means for classifying events, wherein events are classified into categories and selected events are forwarded to said at least one collaborative application based on the category of the event.

7. The management system of claim 6, wherein said event categories comprise:

events due to a user interaction, events which cause an application to redraw itself, events from another application, events which reflect a change in state of said plurality of participants, and events which reflect a change in state of said at least one collaborative application's resources.

8. The management system of claim 7, wherein:

said events due to a user interaction are forwarded only from said input focus participant;

said events due to a user interaction which may trigger a participant switch are forwarded only from a non-input focus participant, and said input focus role is switched to said non-input focus participant sending the event;

said events which cause said at least one collaborative application to redraw itself are sent from all of said plurality of participants;

said events from another application are sent only from said master participant;

said events which reflect a change in the state of said plurality of participants are sent only from said master participant; and

said events which reflect a change in the state of said at least one collaborative application's resources are sent only from said input focus participant.

9. A method for management and classification of requests and their subsequent responses for a multiple participant system including a collaborative application and a non-collaborative application, comprising the steps of:

sending requests asking a participant to draw, or change a state of a participant's resources, from said collaborative application to all participants;

sending requests querying a state of a display from said collaborative application to an input focus participant, and sending any replies to such requests only from the input focus participant to said collaborative application;

sending requests querying a state of a participant from said collaborative application to a master participant, and sending responses to such requests from the master participant to said collaborative application;

sending requests querying or changing a state of input devices from said collaborative application to the input focus participant, so as to permit the manipulation and execution of said non-collaborative application by all other participants and sending responses to such requests from the input focus participant to said collaborative application; and

sending requests initializing resource identifiers for said collaborative application from said collaborative application to all participants, and sending responses to such requests from all participants to said collaborative application.

10. The method of claim 9 further comprising the steps of:

sending events, due to a user interaction, to the collaborative application if they are from an input focus participant;

sending events, due to a user interaction, to said collaborative application if they are to switch the input focus to another participant in said conference;

sending events, which cause the collaborative application to redraw itself, to said collaborative application if they are from a participant in the conference,

sending events, from another application to said collaborative application if they are from a master participant;

sending events, which reflect a change in the state of a participant, to said collaborative application if they are from the master participant;

sending events, caused by a change in the state of a participant's resources, to said collaborative application if they are from the input focus participant.

* * * * *